BUSINESS INTELLIGENCE AND DATA ANALYTICS

University of Petra

Institute of Analytics Recognized Programme

# Introduction to Word Embeddings

## 307307 BI Methods

# Converting Words to Numbers

Old Method:- One-Hot Encoding

Each word is represented by setting one dimension to 1 and all the other dimensions to ZEROS.

| id | color |
|---|---|
| 1 | red |
| 2 | blue |
| 3 | green |
| 4 | blue |

**One Hot Encoding** →

| id | color_red | color_blue | color_green |
|---|---|---|---|
| 1 | 1 | 0 | 0 |
| 2 | 0 | 1 | 0 |
| 3 | 0 | 0 | 1 |
| 4 | 0 | 1 | 0 |

Limitations:

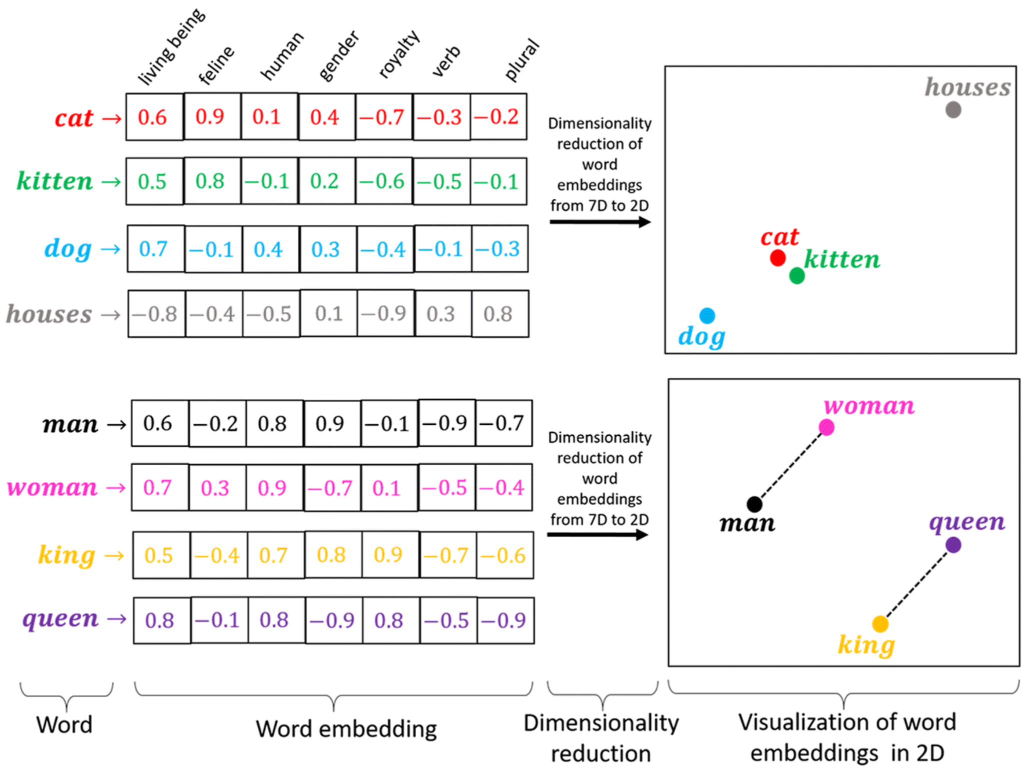
- Vocabulary of 100,000 words → 100,000 dimensions [0010000000000000000000000000000000000000…etc.]
- All words equally distant from each other
- No semantic meaning captured

# What are Word Embeddings?

- Capture semantic relationships

Example:

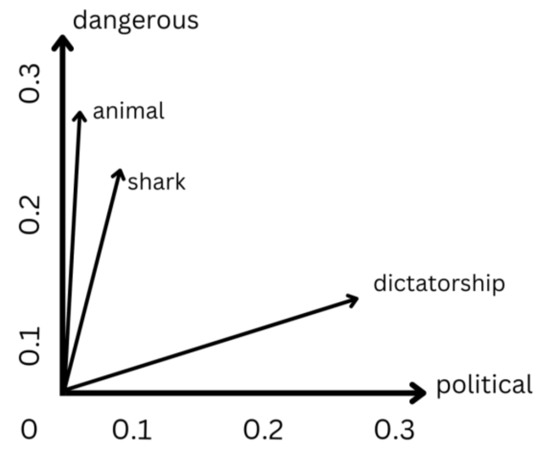
- cat = [0.2, -0.4, 0.7, ..., 0.1] (300 numbers)
- dog = [0.3, -0.3, 0.65, ..., 0.15] (similar to cat!)



# Word Embeddings and Vector Spaces

- Words are encoded as dense numerical vectors instead of one-hot or sparse representations.
- Similar words → similar vectors
- Typically, 50-300 dimensions (vs. vocabulary size in 1-hot-encoding)
- Word Embeddings captures semantic and syntactic relationships about/between words.
- Each dimension potentially captures some syntactic or semantic meaning.
- These vectors are learned from text by models like Word2Vec.

| Word | Dimension 1 (political) | Dimension 2 (dangerous) |
|---|---|---|
| shark | 0.05 | 0.22 |
| animal | 0.03 | 0.25 |
| dangerous | 0.07 | 0.32 |
| political | 0.31 | 0.04 |
| dictatorship | 0.28 | 0.15 |

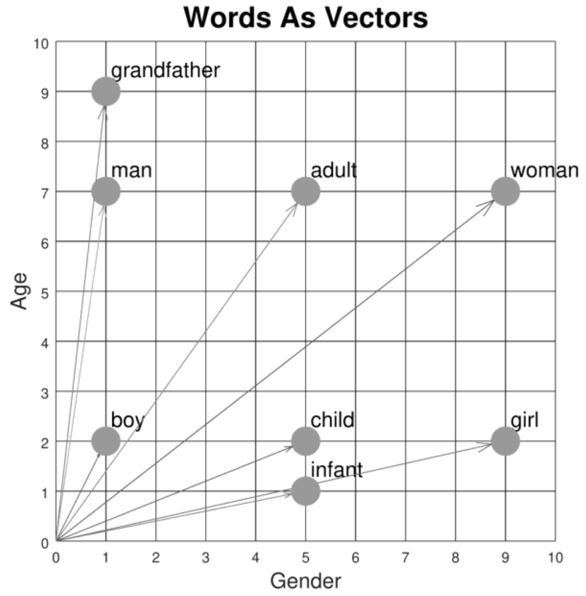


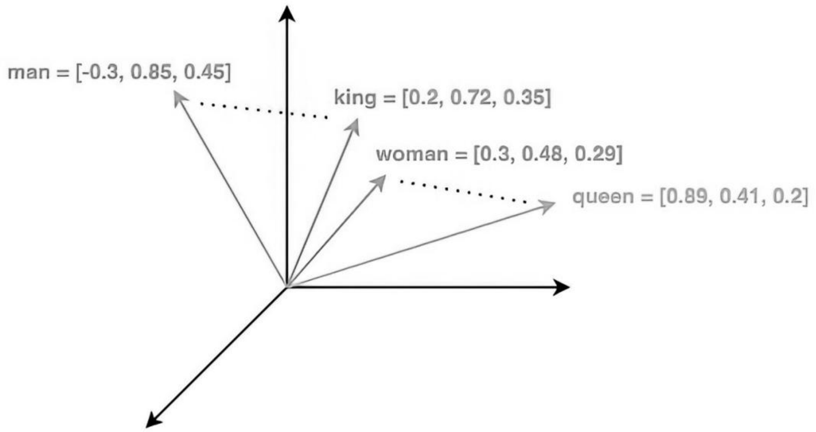
# Vector Spaces and Word Embeddings

- A vector space is a mathematical space where each word is represented as a point (or vector) in multi-dimensional space.
- Makes it possible to compare, visualize, and manipulate meanings of words using math.
- Enables operations like:
  - Similarity: "king" is close to "queen"
  - Analogy: "king" - "man" + "woman" ≈ "queen"

**Properties of Vector Space**

- Semantic relationships are preserved (e.g., "shark" is closer to "dangerous" than "political").
- Similar meanings → closer vectors.
- Dissimilar meanings → vectors farther apart.



# Key Intuition

- Distributional hypothesis - "You shall know a word by the company it keeps" (J.R. Firth, 1957)
- Words in similar contexts have similar meanings:
- "The **cat** sat on the mat"
- "The **dog** sat on the mat"
- "The **rabbit** sat on the mat"
- → cat, dog, rabbit learn similar representations

# Word2Vec (Tomas Mikolov et al., 2013)

Word2Vec  was developed by Tomas Mikolov and team at Google.

Mikolov presented two architectures:

**1) CBOW** (Continuous Bag of Words): Predict word from context

- Input: [the, ___, sat, on] → Output: cat

**2) Skip-gram**: Predict context from word

- Input: cat → Output: [the, sat, on, mat]

- Trained on massive text corpora using simple neural networks

# Mikolov Approach – Convert Text into Input-Output Pairs

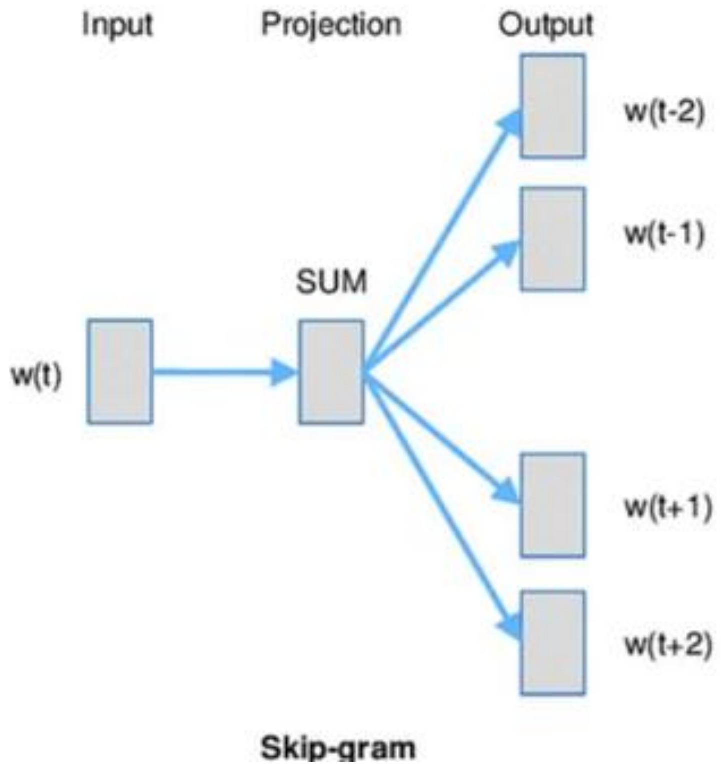
Suppose we have these sentences (corpus), we can convert them into input output pairs to be used for training a neural network.

*"Large language models are transforming business applications",*
*"Natural language processing helps computers understand human language",*
*"Word embeddings capture semantic relationships between words",*
*"Neural networks learn distributed representations of words",*
*"Businesses use language models for various applications",*
*"Customer service can be improved with language technology",*
*"Modern language models require significant computing resources",*
*"Language models can generate human-like text for businesses"*

# Skip-gram Architecture

- **Source Sentence:** "Large language models are transforming business applications"
- **Window Size:** 2 (2 words on each side of target word)
- **Task:** Predict context words from target word
- Given Target Word Predict → Context Words

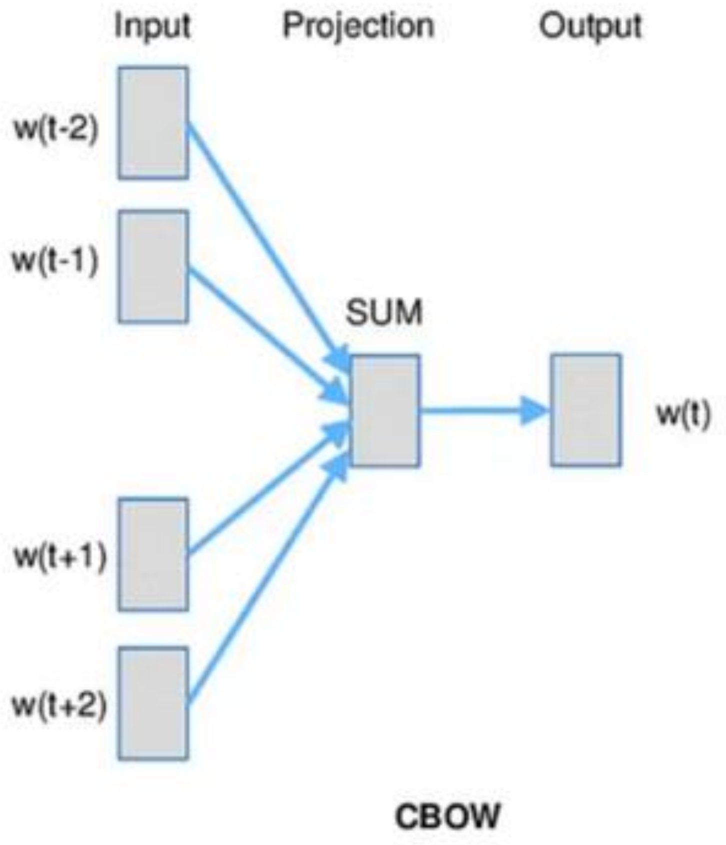
| Input (Target Word) | Output (Context Words) |
|---|---|
| Large | language, models |
| language | Large, models, are |
| models | Large, language, are, transforming |
| are | language, models, transforming, business |
| transforming | models, are, business, applications |
| business | are, transforming, applications |
| applications | transforming, business |



# CBOW (Continuous Bag of Words) Architecture

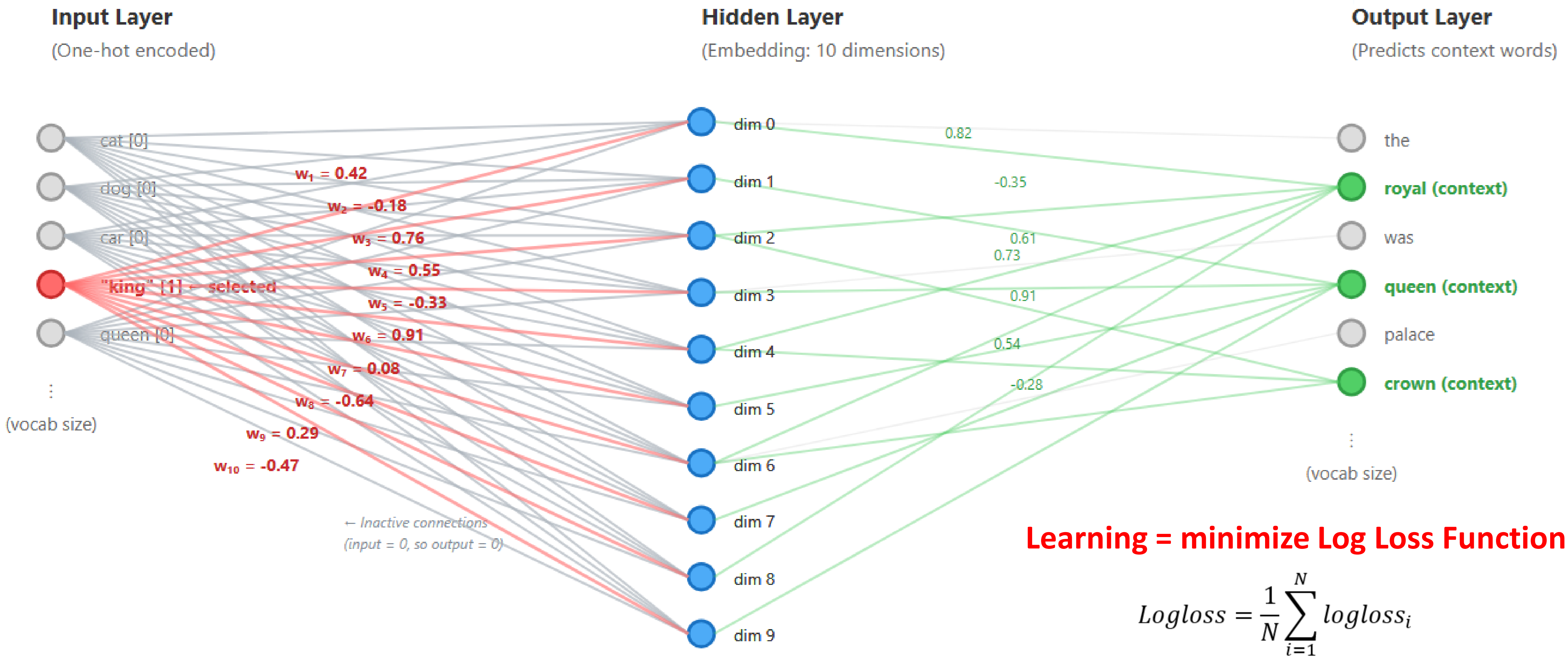
- **Source Sentence:** "Large language models are transforming business applications"
- **Task:** Predict target word from context words
- Given Context Words Predict → Target Word

| Input (Context Words) | Output (Target Word) |
|---|---|
| [language, models] | Large |
| [Large, models, are] | language |
| [Large, language, are, transforming] | models |
| [language, models, transforming, business] | are |
| [models, are, business, applications] | transforming |
| [are, transforming, applications] | business |
| [transforming, business] | applications |



# Word2Vec Skip-gram: Learning Word Embeddings

Training sentence: "The **king** wore a royal crown, and the queen stood beside him"



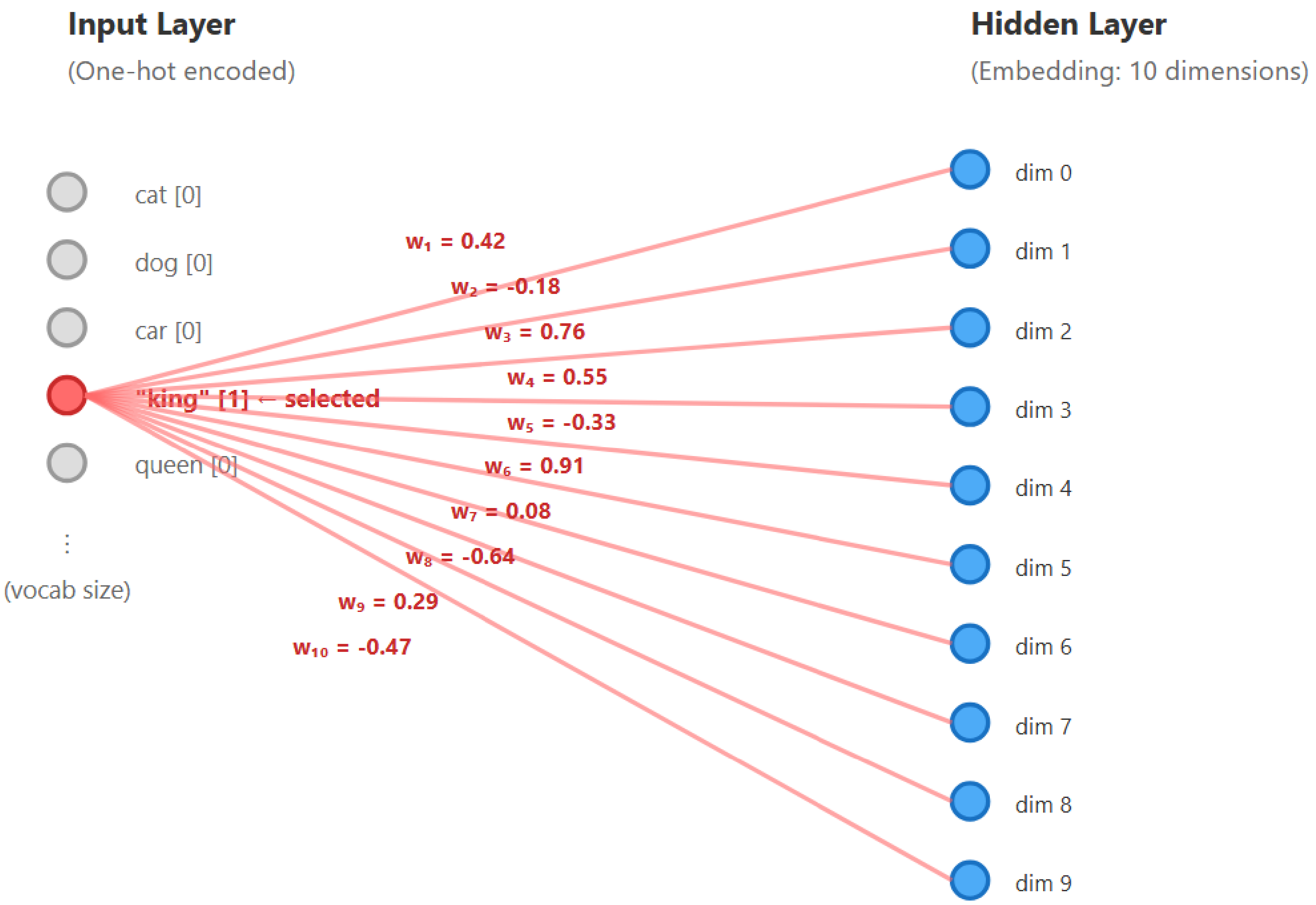

**Learning = minimize Log Loss Function**

$$Logloss = \frac{1}{N}\sum_{i=1}^{N} logloss_i$$

**Training Process: Input Word "king" → Embedding → Predict Context Words**

- Input→Hidden weights = Word Embeddings (e.g., king = [0.42, -0.18, 0.76, 0.55, -0.33, 0.91, 0.08, -0.64, 0.29, -0.47])
- Hidden→Output weights predict context: Given "king", maximize probability of "royal", "queen", "crown"
- Backpropagation updates both weight matrices → words in similar contexts get similar embeddings

# Skip-Gram Model



# Core Math Behind Word2Vec: Softmax & Log-Loss

**1. Objective**

Train word embeddings so that words appearing in similar contexts have similar vector representations.

---

**2. Softmax Function (Prediction Probability)**

For a given *target word* $w_o$ and *input word* $w_i$, the probability of $w_o$ given $w_i$ is:

$$P(w_o|w_i) = \frac{e^{v'_{w_o} \cdot v_{w_i}}}{\sum_{w=1}^{V} e^{v'_w \cdot v_{w_i}}}$$

- $v_{w_i}$: embedding of the input (center) word
- $v'_{w_o}$: embedding of the output (context) word
- $V$: vocabulary size

---

**3. Log-Loss (Objective to Minimize)**

$$L = -\log P(w_o|w_i)$$

Over the whole corpus:

$$J = -\sum_{(w_i, w_o) \in D} \log P(w_o|w_i)$$

---

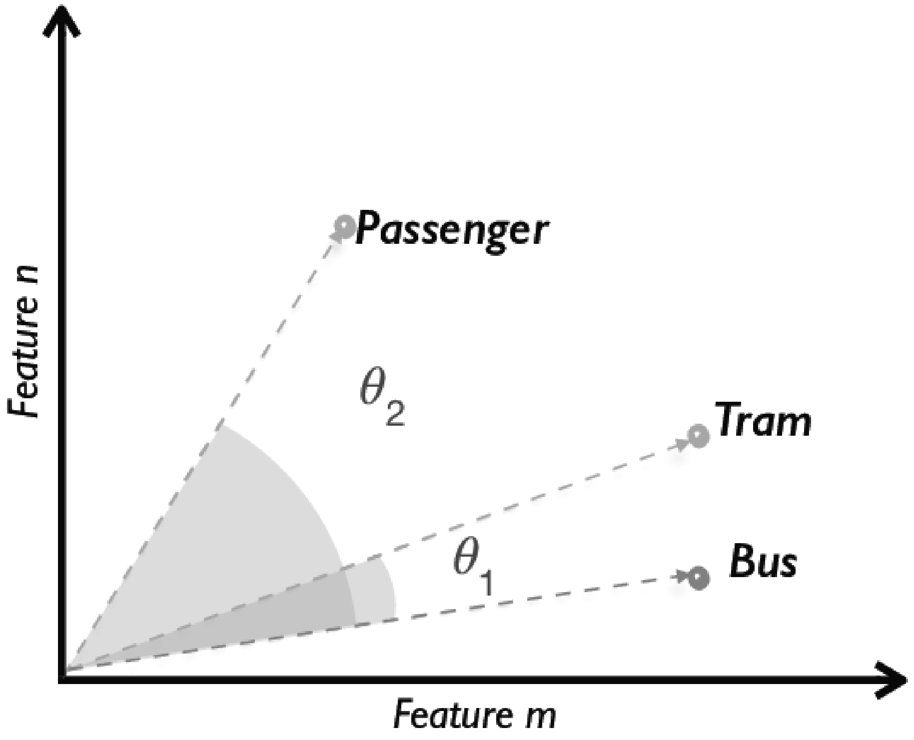
**4. Intuition**

- The **softmax** converts similarity scores (dot products) into probabilities.
- The **log-loss** penalizes incorrect predictions, pushing the correct context words closer in embedding space.

# Measuring Similarity Between Word Vectors

**Why Compare Word Vectors?**

- Word embeddings map words into a vector space.
- **Words with similar meanings** are placed **close together** in that space.
- To quantify this "closeness," we use **vector similarity**.




**Cosine Similarity**

Most common metric used to compare word vectors:

$$\text{cosine\_similarity}(\vec{A}, \vec{B}) = \frac{\vec{A} \cdot \vec{B}}{\|\vec{A}\| \|\vec{B}\|}$$

- Measures the **angle** between two vectors (not their magnitude).
- Ranges from **-1 to 1**:
  - 1 → Same direction (very similar)
  - 0 → Orthogonal (unrelated)
  - -1 → Opposite directions (very different)

**Intuition**

- Vectors for `"king"` and `"queen"` will have high cosine similarity.
- Vectors for `"apple"` and `"keyboard"` will have low similarity.

# Use Pre-Trained Embeddings

Gensim is an open-source Python library used for **topic modeling** and **natural language processing (NLP)**.
It's great for working with **large text datasets** because it doesn't need to load all the data into memory at once.

Key features:

- Builds and uses **word embeddings** (like Word2Vec, FastText, Doc2Vec).
- Measures **semantic similarity** between words or documents.
- **Efficient and memory-friendly**, ideal for handling big collections of text.

```python
import gensim.downloader as api
from gensim.models import Word2Vec

# Load pre-trained Word2Vec model
word2vec_model = api.load("word2vec-google-news-300")


# Get vector for a word
cat_vector = model.wv['cat']
print("Vector for 'cat':", cat_vector[:5])  # Show first 5 dimensions


# Find similar words
similar_words = word2vec_model.most_similar('computer', topn=5)
print("Words similar to 'computer':", similar_words)


# Word analogies
result = word2vec_model.most_similar(positive=['woman', 'king'], negative=['man'], topn=1)
print("king - man + woman =", result)
```

Open in Colab

# Applications of Word Embeddings in NLP

1. **Semantic Similarity**
   Measure how similar two words, phrases, or documents are by comparing their vector representations.
   Example: Identifying that "doctor" and "physician" are closely related.
2. **Text Classification**
   Used as input features for tasks like spam detection, sentiment analysis, and topic classification.
   Embeddings provide rich, dense input for machine learning models.
3. **Named Entity Recognition (NER)**
   Help identify proper nouns and classify them into categories like person, location, or organization.
   Embedding-based models improve contextual understanding of named entities.
4. **Machine Translation**
   Map words from one language to another by aligning embeddings in multilingual space.
   Improves translation accuracy by leveraging semantic proximity.
5. **Question Answering & Chatbots**
   Used to understand queries and match them with appropriate answers or responses.
   Enable bots to interpret intent and context more accurately.

- **6. Information Retrieval**
  Enhance search engines by retrieving results based on semantic meaning, not just keyword matches.
  Example: Searching for “heart attack” returns documents containing “cardiac arrest.”